

**LAPORAN PRAKTIKUM STRUKTUR DATA DAN  
PEMROGRAGAMAN**

**MODUL IX  
GRAPH DAN TREE**



**Disusun oleh :**

ZEFANYA.B.T.TARIGAN

2311102028

IF-11-A

**Dosen Pengampu :**

Wahyu Andi Saputra, S. Pd., M. Eng

PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2024

## BAB I

### A. TUJUAN PRAKTIKUM

- Mahasiswa diharapkan mampu memahami graph dan tree
- Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

## BAB II

### B. DASAR TEORI

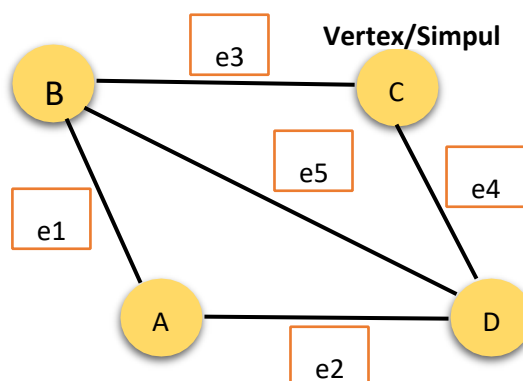
#### 1. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde.

Dapat digambarkan:

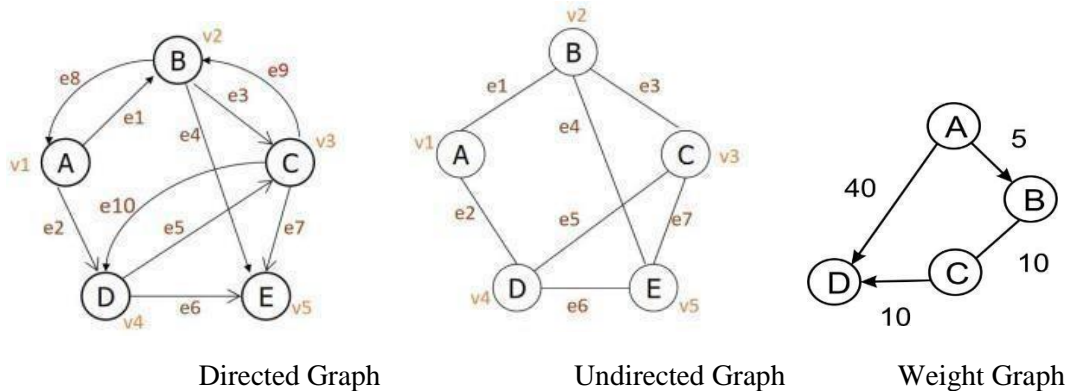


Gambar 1 Contoh Graph

## Edge/Busur

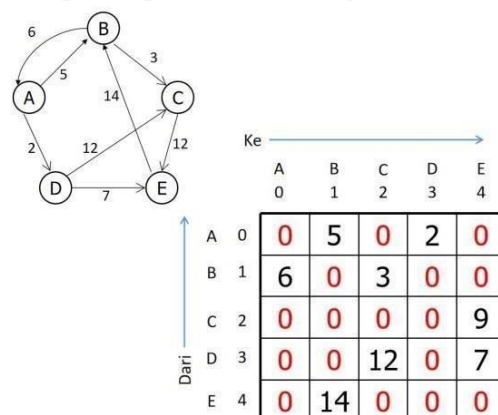
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

### Jenis-jenis Graph



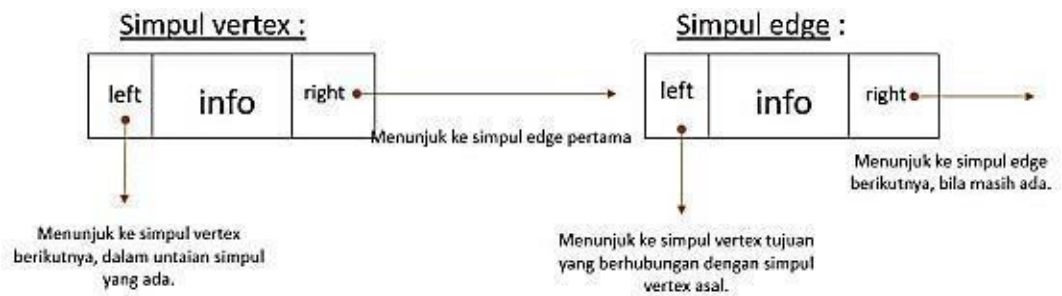
- Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
- Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
- Weight Graph :** Graph yang mempunyai nilai pada tiap edgenya.

### Representasi Graph Representasi dengan Matriks



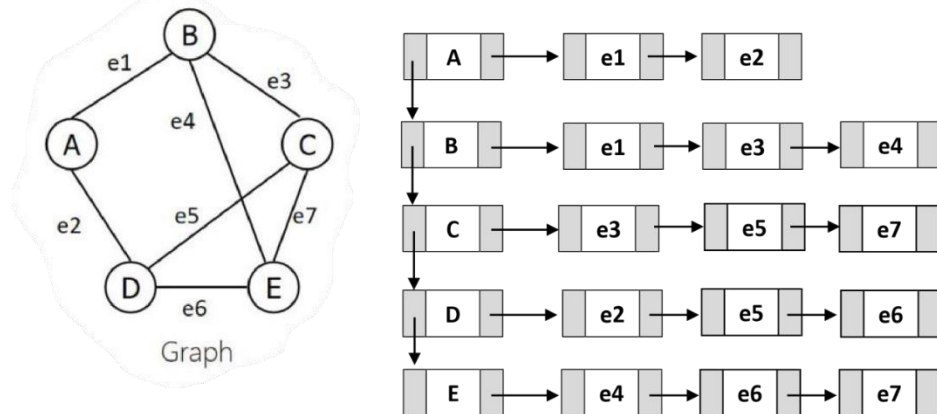
Gambar 4 Representasi Graph dengan Matriks

## Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

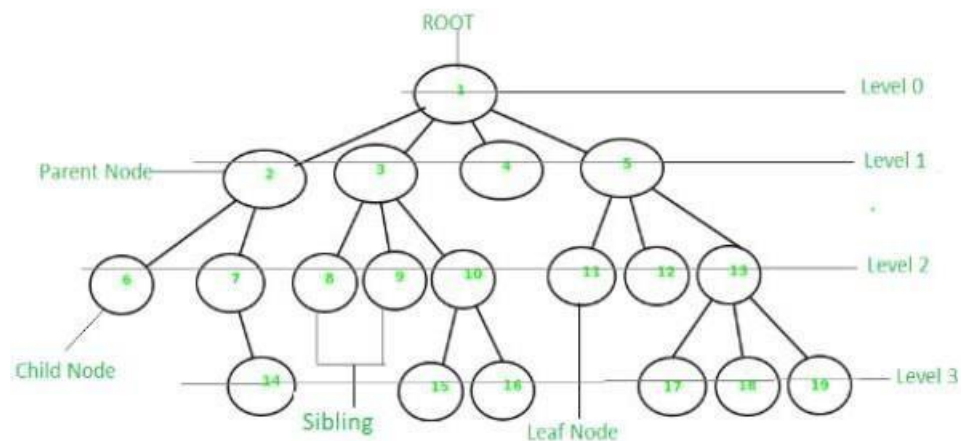
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.



Gambar 6 Representasi Graph dengan Linked List

## 2. Tree atau Pohon

Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :

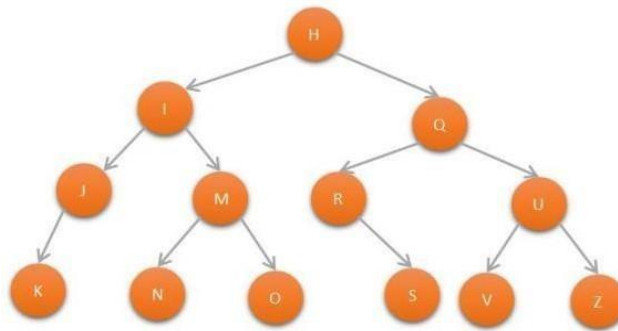


<b>Predecessor</b>	Node yang berada di atas node tertentu
<b>Successor</b>	Node yang berada di bawah node tertentu
<b>Ancestor</b>	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
<b>Descendent</b>	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
<b>Parent</b>	Predecessor satu level di atas suatu node
<b>Child</b>	Successor satu level di bawah suatu node
<b>Sibling</b>	Node-node yang memiliki parent yang sama
<b>Subtree</b>	Suatu node beserta descendent-nya
<b>Size</b>	Banyaknya node dalam suatu tree
<b>Height</b>	Banyaknya tingkatan/level dalam suatu tree
<b>Roof</b>	Node khusus yang tidak memiliki predecessor
<b>Leaf</b>	Node-node dalam tree yang tidak memiliki successor
<b>Degree</b>	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.

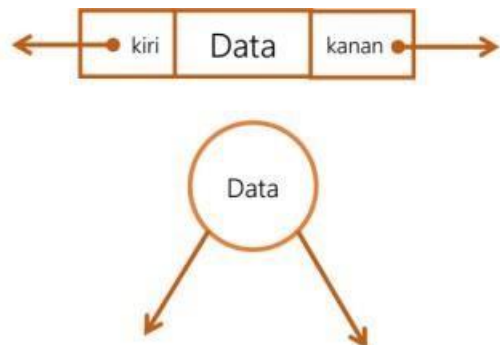
Gambar 1, menunjukkan contoh dari struktur data binary tree.



Gambar 1 Struktur Data Binary Tree

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{  
    char data;  
    pohon *kanan;  
    pohon *kiri;  
};  
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

## Operasi pada Tree

- a. **Create:** digunakan untuk membentuk binary tree baru yang masih kosong.

- b. Clear:** digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- c. isEmpty:** digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- d. Insert:** digunakan untuk memasukkan sebuah node kedalam tree.
- e. Find:** digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
- f. Update:** digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
- g. Retrive:** digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- h. Delete Sub:** digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- i. Characteristic:** digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
- j. Traverse:** digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

### 1. Pre-Order

Penelusuran secara pre-order

memiliki alur: **a.** Cetak data

pada simpul root

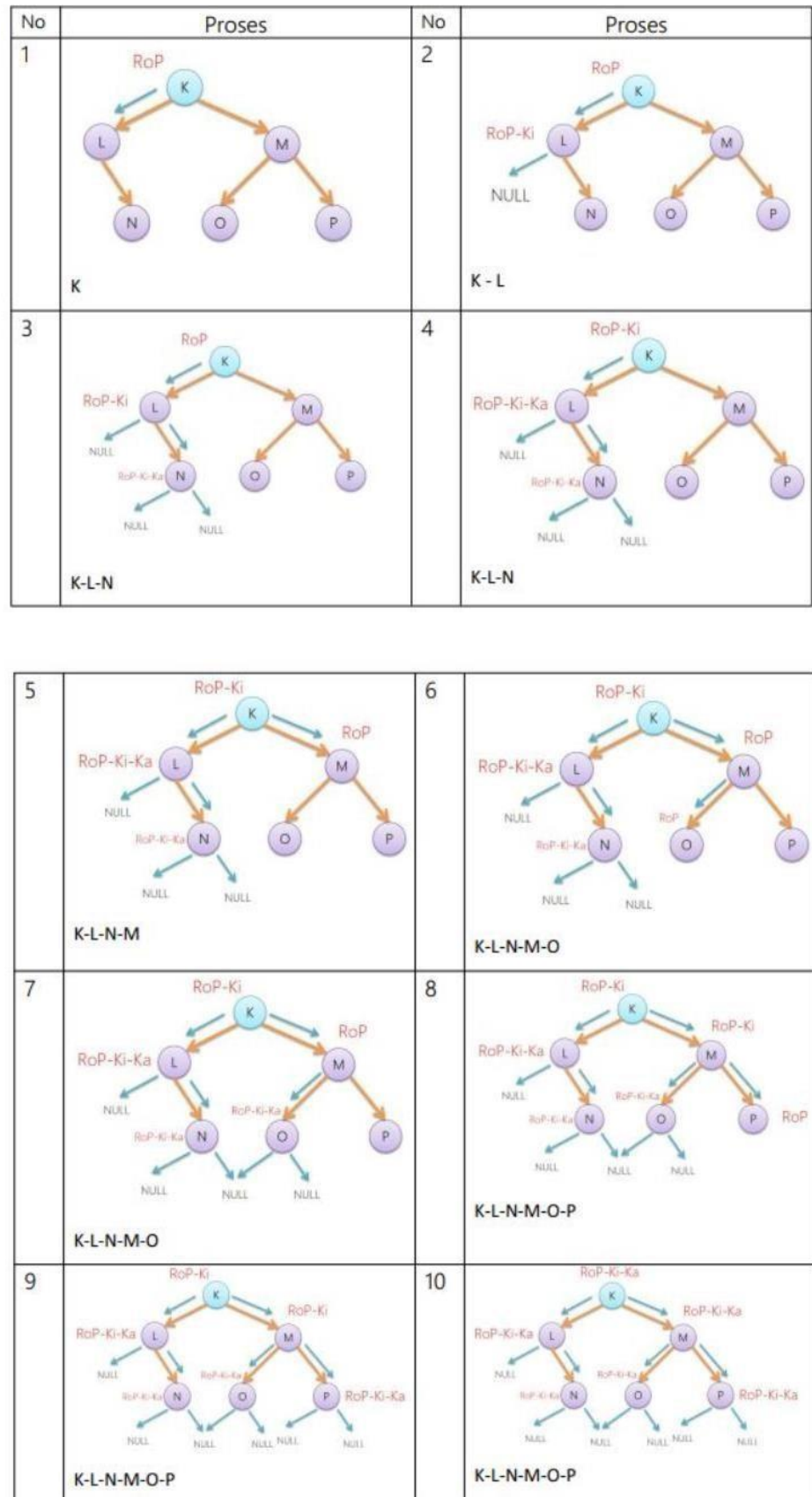
**b.** Secara rekursif mencetak seluruh data pada subpohon kiri

**c.** Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

## Alur pre-order





## 2. In-Order

Penelusuran secara in-order memiliki alur:

- a. Secara rekursif mencetak seluruh data pada subpohon kiri
  - b. Cetak data pada root
  - c. Secara rekursif mencetak seluruh data pada subpohon kanan
- Dapat kita turunkan rumus penelusuran menjadi:

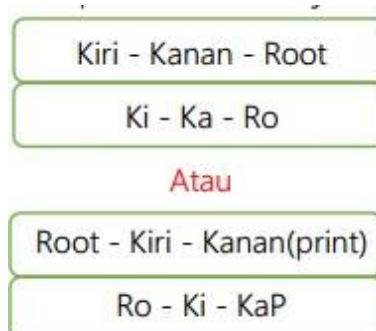


## 3. Post Order

Penelusuran secara in-order memiliki alur:

- a. Secara rekursif mencetak seluruh data pada subpohon kiri
- b. Secara rekursif mencetak seluruh data pada subpohon kanan
- c. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



## BAB III

### GUIDED 1

#### Source code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogjakarta"};

int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15)
              << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
```

```

{
    cout << " " << simpul[kolom] << "(" <<
busur[baris][kolom] << ")";

    }

    }

    cout << endl;

}

}

int main()
{
    tampilGraph();

    return 0;

}

```

## OUTPUT

```

MIEngine-Out-tswgyxq2.jz1' '--stderr=Microsoft-MIEngine-Error-4yf1
'--pid=Microsoft-MIEngine-Pid-iwqyq12y.dw5' '--dbgExe=D:\MinGW\bin\
'--interpreter=mi'
PS D:\KULIAH\SEMESTER2\struktur data praktikum> ./G1m9
Ciamis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS D:\KULIAH\SEMESTER2\struktur data praktikum>

```

## Deskripsi Program

Program ini menunjukkan cara dasar untuk merepresentasikan dan menampilkan sebuah graph menggunakan matriks ketetanggaan dalam C++. Ini sangat berguna untuk memodelkan dan mengelola data yang berbentuk jaringan atau hubungan antar entitas dalam berbagai aplikasi.

## GUIDED 2

### Source code

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
void init()
{
    root = NULL;
}
// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1;
    else
        return 0;
    // true
    // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
```

```

root = new Pohon();

    root->data = data;
    root->left = NULL;
    root->right = NULL;
    root->parent = NULL;

    cout << "\n Node " << data << " berhasil dibuat menjadi
root."

        << endl;

    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada
child kiri!"

                << endl;

            return NULL;

```

```

    }

    else
    {
        // kalau tidak ada
        baru = new Pohon();
        baru->data = data;
        baru->left = NULL;
        baru->right = NULL;
        baru->parent = node;
        node->left = baru;

        cout << "\n Node " << data << " berhasil
ditambahkan ke child kiri " << baru->parent->data << endl;

        return baru;
    }
}

// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kanan!"

```

```

<< endl;

        return NULL;
    }
    else
    {
        // kalau tidak ada
        baru = new Pohon();
        baru->data = data;
        baru->left = NULL;
        baru->right = NULL;
        baru->parent = node;
        node->right = baru;

        cout << "\n Node " << data << " berhasil
ditambahkan ke child kanan " << baru->parent->data << endl;
        return baru;
    }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!!" <<
endl;
        else
        {

```

```

char temp = node->data;

        node->data = data;

        cout << "\n Node " << temp << " berhasil diubah
menjadi " << data << endl;

    }

}

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
}

```



```

else
{
    if (!node)
        cout << "\n Node yang ditunjuk tidak ada!" << endl;
    else
    {
        cout << "\n Data Node : " << node->data << endl;
        cout << " Root : " << root->data << endl;
        if (!node->parent)
            cout << " Parent : (tidak punya parent)" <<
endl;
        else
            cout << " Parent : " << node->parent->data <<
endl;
        if (node->parent != NULL && node->parent->left !=
node &&
            node->parent->right == node)
            cout << " Sibling : " << node->parent->left-
>data << endl;
        else if (node->parent != NULL && node->parent-
>right != node &&
            node->parent->left == node)
            cout << " Sibling : " << node->parent->right-
>data << endl;
        else
            cout << " Sibling : (tidak punya sibling)" <<
endl;
        if (!node->left)
            cout << " Child Kiri : (tidak punya Child
kiri)" << endl;
        else
            cout << " Child Kiri : " << node->left->data <<
endl;
        if (!node->right)
            cout << " Child Kanan : (tidak punya Child
kanan)" << endl;
        else
            cout << " Child Kanan : " << node->right->data
<< endl;
    }
}

```

```

}
// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}
// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

```

```

}
}
// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
        }
    }
}

```

```

    }

    deleteTree(node->left);
    deleteTree(node->right);
    if (node == root)
    {
        delete root;
        root = NULL;
    }
    else
    {
        delete node;
    }
}

}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);

        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Hapus Tree
void clear()
{

```

```

{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node = root)

```

```

{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree
void charateristic()
{

```

```

cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() <<
endl;
}
int main()
{
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG,
*nodeH, *nodeI, *nodeJ;

    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
        << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
        << endl;
    cout << " PostOrder :" << endl;

```

```

postOrder(root);

    cout << "\n"
        << endl;

    charateristic();

    deleteSub(nodeE);

    cout << "\n PreOrder :" << endl;

    preOrder();

    cout << "\n"
        << endl;

    charateristic();

}

```

## OUTPUT

```

C:\... '--interpreter=mi'
PS D:\KULIAH\SEMESTER2\struktur data praktikum> ./g2m9Tree

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kanan G

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

```



```
Data node : C

Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)

PreOrder :
A, B, D, E, G, I, J, H, C, F,

InOrder :
D, B, I, G, J, E, H, A, F, C,

PostOrder :
D, I, J, G, H, E, B, F, C, A,

Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.
```

```
PreOrder :
A, B, D, E, C, F,

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS D:\KULIAH\SEMESTER2\struktur data praktikum> |
```

## Deskripsi Program

kode ini memberikan gambaran yang komprehensif tentang bagaimana pohon biner dapat dibangun, dimanipulasi, dan dihapus dalam bahasa pemrograman C++. Dengan berbagai fungsi yang diimplementasikan, kode ini memungkinkan pengguna untuk melakukan berbagai operasi dasar dan lanjutan pada struktur pohon biner.

## BAB IV

### UNGUIDED 1

#### Source code

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main() {
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> jumlah_simpul;

    vector<string>zefanyabranatertiustarigan_2311102028(jumlah_simpul);

    vector<vector<int>> bobot(jumlah_simpul,
vector<int>(jumlah_simpul));

    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << "Silahkan masukkan nama simpul " << i + 1 << "
: ";
        cin >> zefanyabranatertiustarigan_2311102028[i];
    }
    cout << "Silahkan masukkan bobot antar simpul\n";

    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << zefanyabranatertiustarigan_2311102028[i] <<
"-->" << zefanyabranatertiustarigan_2311102028[j] << " : ";
            cin >> bobot[i][j];
        }
    }
}
```

```

cout << "Silahkan masukkan bobot antar simpul\n";

    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << zefanyabranatertiustarigan_2311102028[i] <<
"-->" << zefanyabranatertiustarigan_2311102028[j] << " : ";
            cin >> bobot[i][j];
        }
    }

    cout << "\n\t";
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << zefanyabranatertiustarigan_2311102028[i] <<
"\t";
    }

    cout << "\n";

    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << zefanyabranatertiustarigan_2311102028[i] <<
"\t";
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << "\n";
    }

    return 0;
}

```

## OUTPUT

```

PS D:\KULIAH\SEMESTER2\struktur data praktikum> ./Ungguided1m9
Silahkan masukkan jumlah simpul : 2
Silahkan masukkan nama simpul 1 : MEDAN
Silahkan masukkan nama simpul 2 : JERMAN
Silahkan masukkan bobot antar simpul
MEDAN-->MEDAN : 3
MEDAN-->JERMAN : 2
JERMAN-->MEDAN : 6
JERMAN-->JERMAN : 9

      MEDAN  JERMAN
MEDAN   3    2
JERMAN  6    9
PS D:\KULIAH\SEMESTER2\struktur data praktikum>

```

## Deskripsi Program

Fungsi `binarySearch` Fungsi ini melakukan pencarian biner (binary search) pada string yang sudah diurutkan. Parameter `const string& sentence` merujuk pada kalimat yang akan dicari. Parameter `char target` adalah karakter yang akan dicari dalam kalimat. Menggunakan dua indeks, `left` dan `right`, untuk menandai batas pencarian.

## UNGUIDED 2

### Source Code

```
#include <iostream>
#include <vector>
using namespace std;

// Declaring the Tree structure
struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root = nullptr;

// Initialize the tree
void init() {
    root = NULL;
}

// Check if the tree is empty
int isEmpty() {
    return (root == NULL) ? 1 : 0;
}

// Create a new node
```

```

Pohon* buatNode(char data) {
    Pohon* newNode = new Pohon();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->parent = NULL;

    // cout << "\nNode " << data << " berhasil dibuat." <<
endl;

    return newNode;
}

// Insert a node to the left
Pohon* insertLeft(Pohon* parent, Pohon* child) {
    if (isEmpty() == 1) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->left != NULL) {
            cout << "\nNode " << parent->left->data << " sudah
ada child kiri!" << endl;
            return NULL;
        } else {
            child->parent = parent;
            parent->left = child;

            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kiri " << child->parent->data << endl;

            return child;
        }
    }
}

// Insert a node to the right
Pohon* insertRight(Pohon* parent, Pohon* child) {
    if (root == NULL) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
}

```

```

} else {
    if (parent->right != NULL) {
        cout << "\nNode " << parent->right->data << " sudah
ada child kanan!" << endl;
        return NULL;
    } else {
        child->parent = parent;
        parent->right = child;
        // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kanan " << child->parent->data << endl;
        return child;
    }
}

// Update node data
void update(char data, Pohon *node) {
    if (isEmpty() == 1) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" <<
endl;
        else {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah
menjadi " << data << endl;
        }
    }
}

// Retrieve node data
void retrieve(Pohon *node) {

```

```

if (!root) {
    cout << "\nBuat tree terlebih dahulu!" << endl;
} else {
    if (!node)
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
    else {
        cout << "\nData node : " << node->data << endl;
    }
}
}

// Find node and display its properties
void find(Pohon *node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->data << endl;
            if (!node->parent)
                cout << "Parent : (tidak punya parent)" <<
endl;
            else
                cout << "Parent : " << node->parent->data <<
endl;
            if (node->parent != NULL && node->parent->left !=
node && node->parent->right == node)
                cout << "Sibling : " << node->parent->left-
>data << endl;
            else if (node->parent != NULL && node->parent-
>right != node && node->parent->left == node)
                cout << "Sibling : " << node->parent->right-
>data << endl;
            else

```

```

cout << "Sibling : (tidak punya sibling)" << endl;
        if (!node->left)
            cout << "Child Kiri : (tidak punya Child kiri)"
<< endl;
        else
            cout << "Child Kiri : " << node->left->data <<
endl;
            if (!node->right)
                cout << "Child Kanan : (tidak punya Child
kanan)" << endl;
            else
                cout << "Child Kanan : " << node->right->data
<< endl;
        }
    }
}

// Pre-order traversal
void preOrder(Pohon *node) {
    if (node != NULL) {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

// In-order traversal
void inOrder(Pohon *node) {
    if (node != NULL) {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}

```



```

// Post-order traversal
void postOrder(Pohon *node) {
    if (node != NULL) {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Delete the entire tree
void deleteTree(Pohon *node) {
    if (node != NULL) {
        if (node != root) {
            node->parent->left = NULL;
            node->parent->right = NULL;
        }
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root) {
            delete root;
            root = NULL;
        } else {
            delete node;
        }
    }
}

// Delete a subtree
void deleteSub(Pohon *node) {
    if (!root)
        cout << "\nBuat tree terlebih dahulu!" << endl;
}

```

```

else {
    deleteTree(node->left);
    deleteTree(node->right);
    cout << "\nNode subtree " << node->data << " berhasil
dihapus." << endl;
}
}

// Clear the entire tree
void clear() {
    if (!root)
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Get the size of the tree
int size(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}

// Get the height of the tree
int height(Pohon *node) {
    if (node == NULL) {
        return 0;
    }
}

```

```

    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return (heightKiri >= heightKanan) ? heightKiri + 1 :
heightKanan + 1;
    }
}

// Display tree characteristics
void charateristic() {
    cout << "\nSize Tree : " << size(root) << endl;
    cout << "Height Tree : " << height(root) << endl;
    cout << "Average Node of Tree : " << (size(root) /
(float)height(root)) << endl;
}

int main() {
    root = buatNode('A');
    int menu, part, part2;
    char zefanyabranatertiustarigan_2311102028;

    vector<Pohon*> nodes;
    nodes.push_back(buatNode('B'));
    nodes.push_back(buatNode('C'));
    nodes.push_back(buatNode('D'));
    nodes.push_back(buatNode('E'));
    nodes.push_back(buatNode('F'));
    nodes.push_back(buatNode('G'));
    nodes.push_back(buatNode('H'));
    nodes.push_back(buatNode('I'));
    nodes.push_back(buatNode('J'));

```

```

insertLeft(root, nodes[0]);

    insertRight(root, nodes[1]);

    insertLeft(nodes[0], nodes[2]);

    insertRight(nodes[0], nodes[3]);

    insertLeft(nodes[1], nodes[4]);

    insertLeft(nodes[3], nodes[5]);

    insertRight(nodes[3], nodes[6]);

    insertLeft(nodes[5], nodes[7]);

    insertRight(nodes[5], nodes[8]);


do
{
    cout << "\n----- PROGHGRAM GRAPH ----- \n"
    "1. Tambah node\n"
    "2. Tambah di kiri\n"
    "3. Tambah di kanan\n"
    "4. Lihat karakteristik tree\n"
    "5. Lihat isi data tree\n"
    "6. Cari data tree\n"
    "7. Penelurusan (Traversal) preOrder\n"
    "8. Penelurusan (Traversal) inOrder\n"
    "9. Penelurusan (Traversal) postOrder\n"
    "10. Hapus subTree\n"
    "0. KELUAR\n"
    "\nPilih : ";

    cin >> menu;

    cout << "-----Running Command...\n";

    switch (menu) {

        case 1:

            cout << "\n Nama Node (Character) : ";

            cin >> zefanyabranatertiustarigan_2311102028;

```

```

nodes.push_back(buatNode(zefanyabranatertiustarigan_2311102028)
);

        break;
    case 2:
        cout << "\nMasukkan nomor untuk node parent :
";
        cin >> part;
        cout << "\nMasukkan nomor untuk node child : ";
        cin >> part2;
        insertLeft(nodes[part], nodes[part2]);
        break;
    case 3:
        cout << "\nMasukkan nomor untuk node parent :
";
        cin >> part;
        cout << "\nMasukkan nomor untuk node child : ";
        cin >> part2;
        insertRight(nodes[part], nodes[part2]);
        break;
    case 4:
        charateristic();
        break;
    case 5:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        retrieve(nodes[part]);
        break;
    case 6:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        find(nodes[part]);
        break;
    case 7:
        cout << "\nPreOrder : " << endl;

```

```

preOrder(root);

        cout << "\n" << endl;
        break;
    case 8:
        cout << "\nInOrder :" << endl;
        inOrder(root);
        cout << "\n" << endl;
        break;
    case 9:
        cout << "\nPostOrder :" << endl;
        postOrder(root);
        cout << "\n" << endl;
        break;
    case 10:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        deleteSub(nodes[part]);
        break;
    default:
        break;
}
} while (menu != 0);
}

```

## OUTPUT

```
'--interpreter=mi'
PS D:\KULIAH\SEMESTER2\struktur data praktikum> ./Ungguided2m9

----- PROGHRAM GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 4
-----Running Command...

Size Tree : 10
Height Tree : 5
Average Node of Tree : 2
```

## Deskripsi Program

Pohon adalah struktur yang merepresentasikan sebuah simpul dalam pohon biner. Setiap simpul memiliki data (karakter), pointer ke anak kiri (left), anak kanan (right), dan parent (parent). root adalah pointer global yang menunjuk ke akar pohon.

## BAB V

## KESIMPULAN

Setelah melakukan pembelajaran mengenai Queue di Bahasa Pemrograman C++ berikut poin utama yang telah dipelajari : 1. Graf adalah kumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). 2. Pohon adalah suatu cabang ilmu yang memiliki banyak terapan. Banyak sekali struktur yang bisa direpresentasikan dengan pohon, dan banyak masalah yang bisa diselesaikan dengan bantuan pohon.

## DAFTAR PUSTAKA

Trivusi. (16 September, 2022) Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya.  
diakses pada 08 Juni 2024 dari <https://www.trivusi.web.id/2022/07/struktur-data-graph.html>